BERGISCHE UNIVERSITÄT WUPPERTAL

University of Wuppertal
School of Mathematics and Natural Sciences
Dr. Holger Arndt, M. Sc. Martin Galgon

Tools

Summer Term 2016

Exercise Sheet 3

Exercise 8 *(Python, Classes & Objects)* ——— (6 Points)

Write a class that represents a cookie recipe, `CookieRecipe`. A recipe consists of a list of ingredients including the required total amount and a sequence of steps to perform.

a) The single steps of creating cookies are adding an ingredient, kneading the dough, rolling the dough, cutting the dough into single cookies and finally baking the cookies.

For simplicity, let the ingredients be identified by a string, e.g., "Egg", and handle time as simple number. All amounts can be given without unit of measure.

- Add a method `add(self, ingredient, amount)` that allows to add a certain amount of an ingredient to the recipe.
- Add a method `knead(self, time)` that corresponds to kneading the dough for a certain amount of time.
- Add methods `roll(self)` and `cut(self)` with obvious meaning.
- Add a method `bake(self, time)` that represents baking the cookies for an amount of time.

Not all actions make sense in all states of the process. Implement a way to keep track of the state of the recipe, i.e., whether it is raw, kneaded, rolled, cut or baked. Consider the following sensible restrictions.

- An ingredient can only be added if the state is raw or kneaded, but not if the state already advanced to rolled, cut, or baked. Adding an ingredient resets the state to raw.
- The dough can be kneaded always as long as the cookies are not yet baked.
- The dough can be rolled if it is kneaded but not yet cut.
- The dough can be cut if it is rolled but not yet baked.
- The dough can be baked if it is cut but not yet baked.

If a condition for a step is not met when calling the method, your program should issue an insightful error message.

b) Use a dictionary to keep track of the ingredients added by the `add` method and the corresponding *total* amount. Add a `__str__` method to the `CookieRecipe` class that returns the list of ingredients and corresponding total amounts as a properly formatted string.

Hint: *To see if a key exists in a dictionary, use the* ***`in`*** *keyword.*

c) To keep track of the several steps the cookie baking process may take, implement a small class for each of the single steps that holds the required attributes (like time, amount, ingredient or combinations of those) and implements a `__str__` method to format the single step into a string, e.g., "Bake for X minutes.". Use a list to store objects for all steps performed on the recipe in the order in which they were issued. Add the list of steps to the `__str__` method of the `CookieRecipe` class.

You can check the different states of your implementation with the following example.

- Add 10 flour, 5 water, 3 eggs, 0.25 salt and 5 fish-shaped crackers.
- Knead the dough for 10 time units.
- The dough is too thin, add 5 more flour and 0.5 unsaturated polyester resin.
- Knead again for 5 time units, then roll the dough.
- Dough seems a little hard, knead for 2 time units to be able to add one more egg.
- Knead for 2 time units, roll and cut.
- Forgot sugar. Knead for one time unit, add 2 sugar and 10000 chocolate chips.
- Knead for one time unit, roll, cut and bake for 30 time units.

Exercise 9 *(Python, Regular Expressions)* ——— (6 Points)

Design regular expressions for the following patterns.

a) Strings that start with *A*, end with *C*, and at least have one occurrence of *CD* in between, e.g., *ACDC*, *ACDCDC*, etc.

b) Strings that are simple sentences. In our case that shall mean:
   - a sentence consists of words, i.e., sequences of word characters of length ≥ 1, and must at least contain one word,
   - words are separated by one space character,
   - a sentence starts with a word that starts with a capital letter,
   - a sentence ends with one of the terminators *[.!?]*,
   - the terminators directly follow the preceding word,
   - parts of the sentence may additionally be separated by a comma *[,]*, but only in between words in which case the comma directly follows the preceding word.

c) Strings that represent dates, i.e., numbers for day, month, and year, where the days are in $[1, 31]$ with or without leading zero for single-digit numbers, months are in $[1, 12]$ with or without leading zero for single-digit numbers, and years are either two-digit numbers in $[00, 99]$ or four-digit numbers in $[1970, 2069]$. All numbers are separated by *[.]*.

d) All palindromes of word characters up to length seven, e.g, *a*, *aa*, *aba*, ..., *abccba*, *abcdcba*, etc. Your regular expression may match the empty string as well.

Feel free to submit your regular expressions in form of a small Python testing program.

Exercise 10 *(Fortran, Cellular Automata)* ——— (4 Points)

The cellular automaton known as *Wireworld* was proposed by Brian Silverman in 1987 to simulate electronic-like logic circuits. Electrons are simulated as pairs of heads (`H`) and tails (`T`) that move along conductive tracks (blank parts) on a matrix of cells.

```
############################
##   #####   #####       ##
# ### TH   #   HT ####### #
## HT##### # #####TH     ##
##########   ##############
########## ###############
##########T###############
##########H###############
########## ###############
```

Modeling the matrix of cells as two-dimensional character array with heads (`H`), tails (`T`), conductive cells (blank), and non-conductive cells (`#`), implement the simulation in an endless loop, in each step applying a set of rules to the array and printing it to the console, similar to the example above. See the hints for a nicer program flow.

The rules for advancing the simulation to the next step are simple. Note that the rules have to be applied in sequence to the character matrix in order to work properly.

- All tails (`T`) are marked, become (`X`).
- All heads (`H`) become tails (`T`).
- Conductors (blank) become heads (`H`) if one or two cells in the 8-cell neighborhood are now tails (`T`).
- All marked tails (`X`) become conductors (blank).

Apply rules 1 & 2 simultaneously using a `where`-statement. Then apply rule 3 using nested loops. You can assume that a frame of non-conductive cells is always present around the whole area so that you do not have to mind cells at the border of the array (there is also a conductive "exit" in this frame; it is safe to ignore that as well). Apply rule 4 using again a `where`-statement. **Do not use functions or subroutines for now.**

On the lecture's website you can find an incomplete program containing example setups as an array with name `world`.

Hints:

- *You can clear the terminal on Linux systems using* `call system('clear')`.
- *You can let the program sleep for one second on Linux systems using* `call sleep(1)`.
- *To stop the program, use* `Ctrl-C`.

Deadline: Mon., 20.06.2016, 2:00 pm

Send an email to `tools@studs.math.uni-wuppertal.de` with your source code attached.

BERGISCHE UNIVERSITÄT WUPPERTAL

University of Wuppertal
School of Mathematics and Natural Sciences
Dr. Holger Arndt, M. Sc. Martin Galgon

Tools

Summer Term 2016

Exercise Sheet 4

Exercise 11 *(Fortran, Cellular Automata II)* ——— (6 Points)

This exercise extends task 10 from sheet 3.

If you have not solved this exercise, you can find an alternative implementation on the lecture's website. Note that this alternative implementation uses two arrays to apply all rules at once.

a) Replace the explicit creation of the world array in the code by a subroutine that

- accepts parameters `world` and `path`, where `world` is an allocatable array and `path` is a string pointing to an input file,
- opens the file identified by `path` and reads two integers, the width and the height of the `world` array,
- allocates the `world` array to this size,
- reads the file line by line into an intermediate string buffer, then, character by character, into the correct position inside the `world` array,
- and closes the file.

Place the subroutine in the `contains` section of your program.

Hints:

- *Open the input files provided on the lecture's website in a text editor to examine their structure.*
- *You can safely assume that a line in an input file is never longer than 100 characters.*
- *In your main program, use the* `getarg` *extension to pass a path to an input file from the command line to your program (c.f. script sec. 9, p. 9-3).*
- *It is possible to allocate the array inside the subroutine and then use it outside (c.f. script sec. 5.9, p. 5-29).*
- *Do not forget to deallocate the* `world` *array.*

b) Move the code segment counting the neighbors of a world cell (not including the check for the empty cell) to a separate function `rule3` accepting as parameters the `world` array and the current coordinates `i` and `j`. Let the function return the proper character that has to be placed into the `world` array at position (i, j) for the next step, so that you can call your function as

```
world(i,j) = rule3(world, i, j)
```

for the solution from task 10, or as

```
world2(i,j) = rule3(world, i, j)
```

for the alternative version from the lecture's website.

c) Move all functions and subroutines from your program's `contains` section to a module `ww` and move the module to a separate file. Use the module in your program.

Exercise 12 *(Fortran, Soccer Betting Game)* ——————————————— (5 Points)

Write a Fortran module for betting on the results of a soccer championship. The module should contain

- a type `tip` that represents a tip for a single match, consisting of only two integers – the number of goals for each team,
- a type `bet` that represents all tips for a tournament of 12 matches, consisting of a string `name` for the name of the person placing the bet and an array of tips of length 12,
- and a subroutine `evaluate` that compares the tips to the real outcome and prints the results.

Store the real outcome of the tournament in a variable of type `bet` in your main program. Choose an appropriate name for the reference results, e.g, "UEFA Euro 2016". The results can be hardcoded in your program, e.g., $2-1$, $0-1$, $2-1$, $1-1$, $1-0$, $2-0$, $0-1$, $1-0$, $1-1$, $0-2$, $0-2$, and $1-1$.

Let the user enter his name and his tips for each game and store them in a second variable of type `bet`.

Pass the real results and the user's tips to the subroutine which evaluates each match like follows.

- If the final outcome (i.e. the winner or a draw) was predicted correctly, the user gets 3 points for this match. The actual number of goals does not matter here.
- If the exact goal difference was predicted correctly, the user gets 1 point. It does not matter which team won the match.
- If the total number of goals was predicted correctly, the user gets 2 points. Again, the winner does not matter.

The subroutine should also print the names in a header line, the results for each game, and the final summed score for the user. It should be properly formatted, like this:

```
Results UEFA Euro 2016 -- Martin
Match  1:   2 - 0   (2 - 1)   points: 3
Match  2:   1 - 2   (0 - 1)   points: 4
Match  3:   1 - 1   (2 - 1)   points: 0
Match  4:   2 - 1   (1 - 1)   points: 0
Match  5:   2 - 1   (1 - 0)   points: 4
Match  6:   2 - 1   (2 - 0)   points: 3
Match  7:   1 - 0   (0 - 1)   points: 3
Match  8:   2 - 1   (1 - 0)   points: 4
Match  9:   1 - 0   (1 - 1)   points: 0
Match 10:   2 - 1   (0 - 2)   points: 0
Match 11:   2 - 1   (0 - 2)   points: 0
Match 12:   2 - 1   (1 - 1)   points: 0
Sum: 21
```

Each line shows the number of the match, then the user's tip, the actual outcome in parentheses, and the points the user got for this match.

Hint: *Look up the function* `trim` *for printing the name strings.*

You can test your program with the input file `martin.in` from the terminal via

```
yourprogram < martin.in
```

Exercise 13 *(Fortran, Newton's Method & Recursion)* ——————————— (5 Points)

Consider simple polynomials of the form

$$a_n x^n + \ldots + a_2 x^2 + a_1 x + a_0 \qquad a_i \in \mathbb{R} \forall i.$$

In Fortran, these polynomials may be represented by saving their coefficients in an array such that coefficient a_i is stored at position $i + 1$. Assume that n is not greater than 4. Then, e.g., `(/1,2,1,0,0/)` represents $x^2 + 2x + 1$ and `(/1,0,0,1,0/)` represents $x^3 + 1$.

a) Write a function that evaluates a polynomial f at position x.

b) Write a function that differentiates a polynomial f and returns the new polynomial again as an array.

Now consider the bare essentials of Newton's method for finding roots of real functions,

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}.$$

The recursive sequence x_n provides successively better approximations to a root of f, given some initial guess x_0.

c) Write a recursive function to compute x_n with parameters f, the polynomial, df, it's derivative, and x, an initial guess for the root.

d) In order to abort the recursion, keep track of recursion depth using a `save` variable and return the initial guess, if the limit is surpassed.

e) Use an additional optional parameter n to set maximum recursion depth. If n is not supplied, set the limit to 100.

The output of your program should simply be the approximate position of a root of f.

Note that this method may very well fail for certain functions and/or certain initial guesses.

Deadline: Mon., 11.07.2016, 2:00 pm

Send an email to `tools@studs.math.uni-wuppertal.de` with your source code attached.